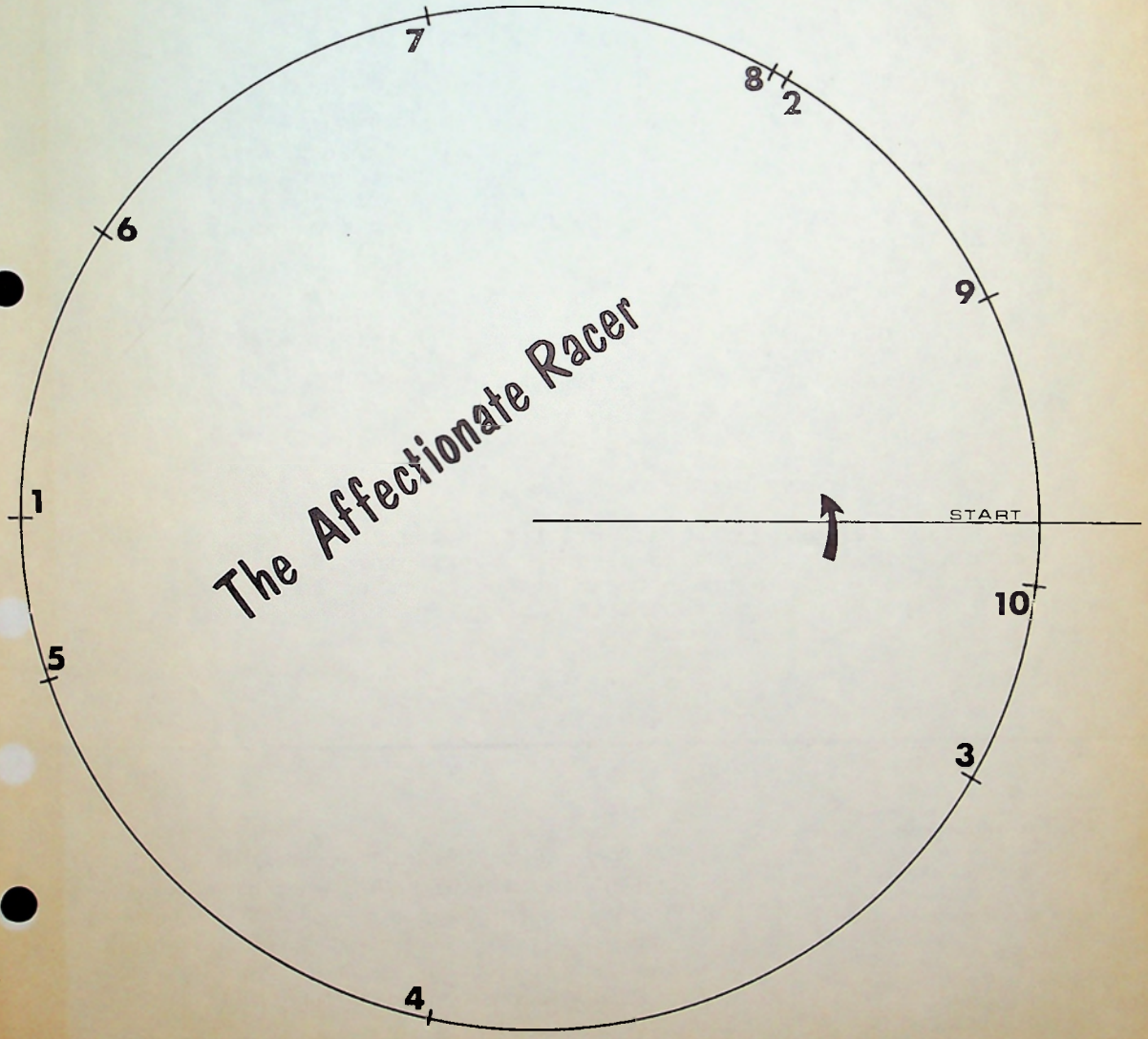
$2.50

62

May 1978 Volume 6 Number 5

Popular Computing

The magazine for people who enjoy computing



The Affectionate Racer

A racing driver travels a circular course, taking the following steps:

1/2 a revolution, then
2/3 revolution,
3/4 revolution,
4/5 revolution,
5/6 revolution,... and so on.

At the end of each step he waves to his girl, at START. As shown by the figure, at the 10th step he is close to her (within 7.156 degrees). At the 29th step, he is even closer. Eventually, he can get as close as he pleases.

The Problem is, at what step will he be within some arbitrary distance of START, say one degree?

Contributing Editor Thomas R. Parkin writes:

"The problem is to sum the series 1/2 + 2/3 + 3/4 +... for some number of terms until the sum is within 1/360 of an integral number of revolutions. This can be done by evaluating a cumulative sum of n/n+1, starting at n = 1, and continuing until the fraction is close enough to an integer.

We could also note that this sum can be expressed as:

$$(1 - 1/2) + (1 - 1/3) + (1 - 1/4) + \ldots = N - \sum_{n=1}^{N}(1/n).$$

The second term of this sum is the sum of the harmonic series and unfortunately there is no simple formula for the sum of N terms of the harmonic series."

The following program, written in the JOSS language, will handle the problem as given. Everything in the program is self-explanatory except the function in step 1.24, "ip." This stands for "integer part."

PROBLEM 229

Continued on page 17...

Publisher: Audrey Gruenberger
Editor: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald
Contributing Editors: Richard Andree
William C. McGee
Thomas R. Parkin
Advertising Manager: Ken W. Sim
Art Director: John G. Scott
Business Manager: Ben Moore

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each.

Contest 15 Result

Contest 15 (Reverse Knockout) in our December issue brought entries from all over the world.

The Problem was this (see Figure 1): Start with all the natural numbers from 3 up. Reverse all sets of 3. This brings 5 to the head of the stream, which dictates knocking out the 5th number after the 5, which is 6; the 5 replaces the number knocked out. The process now repeats. The leading number is 4, so all sets of 4 are reversed, bringing 7 to the front, and the 7 knocks out the 5. The list of numbers that are knocked out (those circled in the diagram) is to be extended.

When the problem was constructed, it was our belief that the only possible solution would involve actual manipulation of huge blocks of numbers; that is, a brute-force approach. This would tend to restrict entries for the contest to those who have access to large and fast machines.

Associate Editor David Babcock wrote a program in assembly language that produced a list of the first 48 numbers, using this crowbar technique; the run took 36 seconds of CPU time. A longer list, using the same approach, would require much storage swapping and overlays and enormous amounts of machine time.

We were pleasantly astonished, then, to find that most entries involved a combined analytic and computer attack on the problem. The material submitted by our contest winner:

John D. Beeby
Millbrae, California

is an outstanding piece of computing, and we are pleased to display most of it verbatim.

But even the entries that chose to manipulate lists of numbers surprised us with their ingenuity, elegance, and sophistication. The best example is from Joan Farmer, of the Polytechnic of North London (the winner of our eleventh contest). It is worthwhile to study her program and her analysis:

"The best solution to this problem is an analytical one. Having failed, as I did, to find such a solution, one is faced with the impossibility--common to this type of problem--of packing an unbounded list of numbers into a finite machine.

"My first approach was based on the fact that we are continually being told that hardware is fast becoming so cheap that we can more or less disregard it, while programming costs are continuing to soar. So I decided to forget about economies of space and time and concentrate on producing a simple, clearly constructed program that would mirror the inherent features of the problem and be easy to understand and maintain (even if it did need a computer with an outrageously large memory to produce a reasonably long list of results).

"The problem revolves 'round two lists--a list of the natural numbers and a list of "leaders"--both of which are continuously growing and need to be permanently remembered. The list of numbers contains the integers in their current state of reversal, and each leader remembers the span of its reversals and how far through the numbers these reversals have been made. No leader may, of course, make a reversal beyond the point reached by its predecessor. Thus, a natural need for recursion arises.

"I decided to hold leaders and numbers as linked lists, the links of the number list being altered to record reversals rather than swapping actual values, and both lists being held in internal memory. The program proved to be extremely simple. It was written in Algol 68 which provides the necessary list processing and recursion. Each new leader reverses his first span of numbers and then continues reversing until his candidate for knocking out is within range. If necessary, he calls on the previous leader to do sufficient reversals to cover the required range. If the recursive calls for reversal get back to the original leader, he extends the list of numbers by adding on reversed groups of 3.

"Only four procedures were required:

extend--to add new numbers to the end of the number list;
reverse--to provide the recursive reversal mechanism;
knockout--to pick off the next number for printing and replace it by the current first number;
newleader--to introduce the next leader to the head of the list.

"There was only one snag with this program. I didn't have the postulated cheap but extensive hardware to run it on and so it used up all the available memory after printing only 41 numbers. I decided to submit my solution and think again!"

knockout pc Dec 77

```
begin
mode number = struct (int n, ref number next);
mode leader = struct (int val,           c length of exchange      c
                      start,             c ordinal of next chain   c
                      ref number last,   c end of last chain       c
                      ref leader next);  c previous leader         c
ref leader nil  = nil;
ref number niln = nil;
ref number firstn :=  number := (2, niln);
ref number lastn  := firstn;
ref leader firstl := leader := (3, 3, firstn, nil);
int a := 3,  z := 2; c ordinals of start and end of integers      c

proc extend = void:
c add 3 more numbers to the number list in reverse order c
begin
   for j from z plus 3 by -1 to z - 2
   do begin
      next of lastn := number := (j, niln);
      lastn := next of lastn
   end
end;   c end of extend c

proc reverse = (ref ref leader l) void:
c reverse the order of the next set of numbers in l's chain       c
begin
   ref number lf, n, p, q, r;
   if next of l is nil
   then (extend; last of l := lastn)
   else while (start of l + val of l) >= start of (next of l)
      do reverse (next of l);
      lf := last of l;
      p := next of lf; q := next of p; r := next of q;
      n := p;  c hang on to the new 'last' c
      to val of l - 2
      do begin
         next of q := p;     c reverse a pointer   c
         p := q; q := r;     c and                 c
         r := next of r      c move on one.        c
      end;
      next of q := p;        c tidy up ends        c
      next of n := r;
      if (ref number val firstn) is next of lf
         then firstn := q fi;
      next of lf := q;
      last of l := n
   fi;
   start of l plus val of l
end;   c end of reverse c
```

```
proc knockout = (ref leader l) void:
c print next candidate and change its value c
begin
   ref number rn := firstn;
   to n of firstn do rn := next of rn;
   print (n of rn);
   if charsleft(standout)   10 then print (newline) fi;
   n of rn := n of firstn
end;   c end of knockout c

proc newleader = void:
c add a leader to the head of the list of leaders c
begin
   firstl := leader := (n of next of firstn, a plus 1, firstn, firstl);
   firstn := next of firstn   c drop the first number c
end;   c end of newleader c

c  the main program starts here  c

print (("Popular Computing contest 15(PC57-2, Dec 77)",
        newline, newline));
reverse (firstl);
firstn := next of firstn;
do begin
   reverse (firstl);
   while a + n of firstn >= start of firstl do reverse (firstl);
   knockout (firstl);
   newleader
end
end
finish
```

But the brute-force approach, however cleverly implemented, can never extend the knockout list very far; after relatively few stages, the procedure would call for reversing blocks of millions of numbers and even with extensive (virtual) storage, the CPU time would become excessive. So the analytic approach must take over. What follows, then, is Mr. Beeby's solution.

In my solution the diagram of Problem 212 (PC57-1) is expanded, figuratively speaking, one "cycle" (that is, one pair of lines) at a time down the page and as far to the right as necessary. Actually, all of the numbers in this vast hypothetical structure are ignored except for three key elements in each cycle. One of these is the extraction or output item. The others (described in Figure 2 as "starter" and "pointer") are stored for use by the program. Hence the modest storage requirement is two items per output line. The method by which each set of three key elements is determined is explained in detail in the discussion which follows.

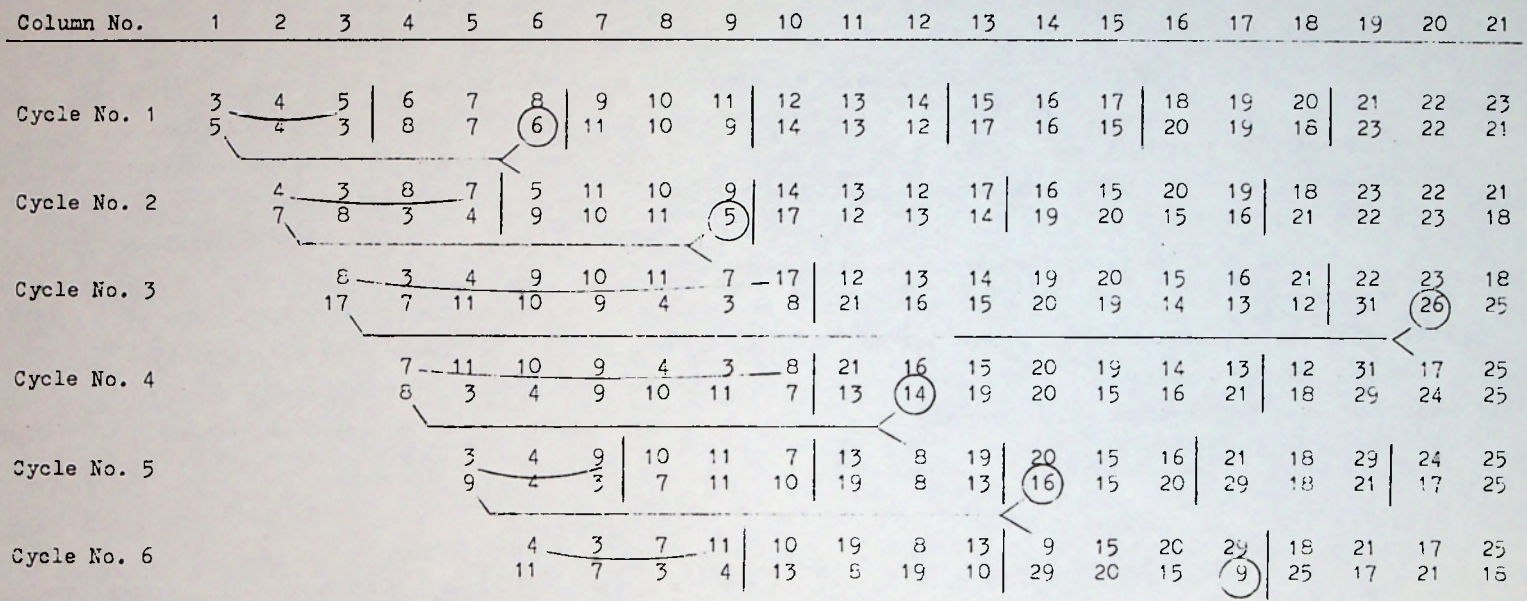


Figure 1. This is the diagram which appeared on the cover of Popular Computing for December, 1977. We have added column numbers and what we have chosen to call cycle numbers, a cycle referring to a pair of lines on the diagram. The entire diagram (or the chart, as we usually speak of it) may be pictured as extending several million columns to the right and several hundred cycles down the page. One important fact about the chart is that each number on the top line is exactly two plus its column number.

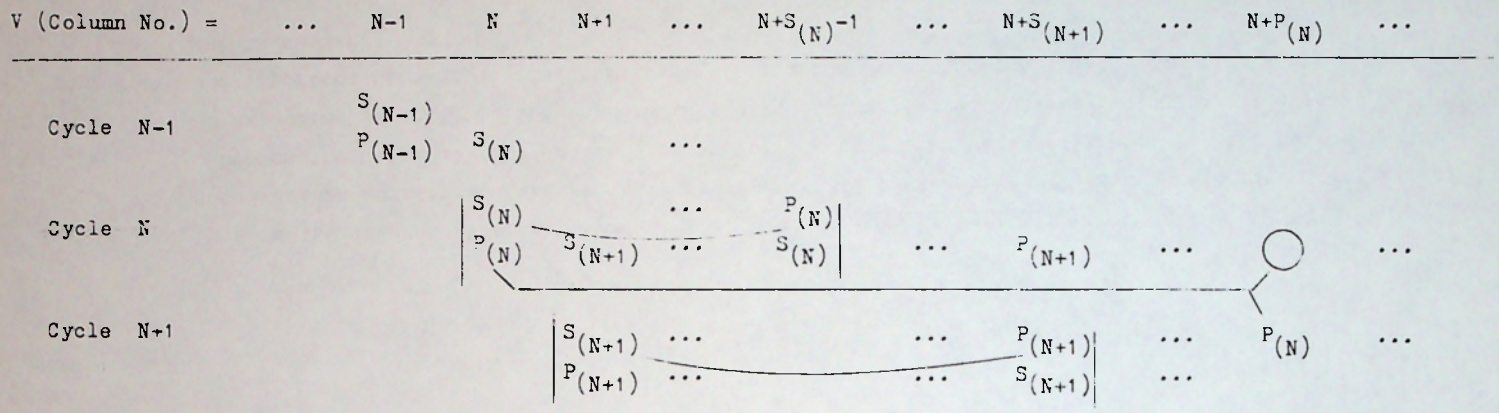
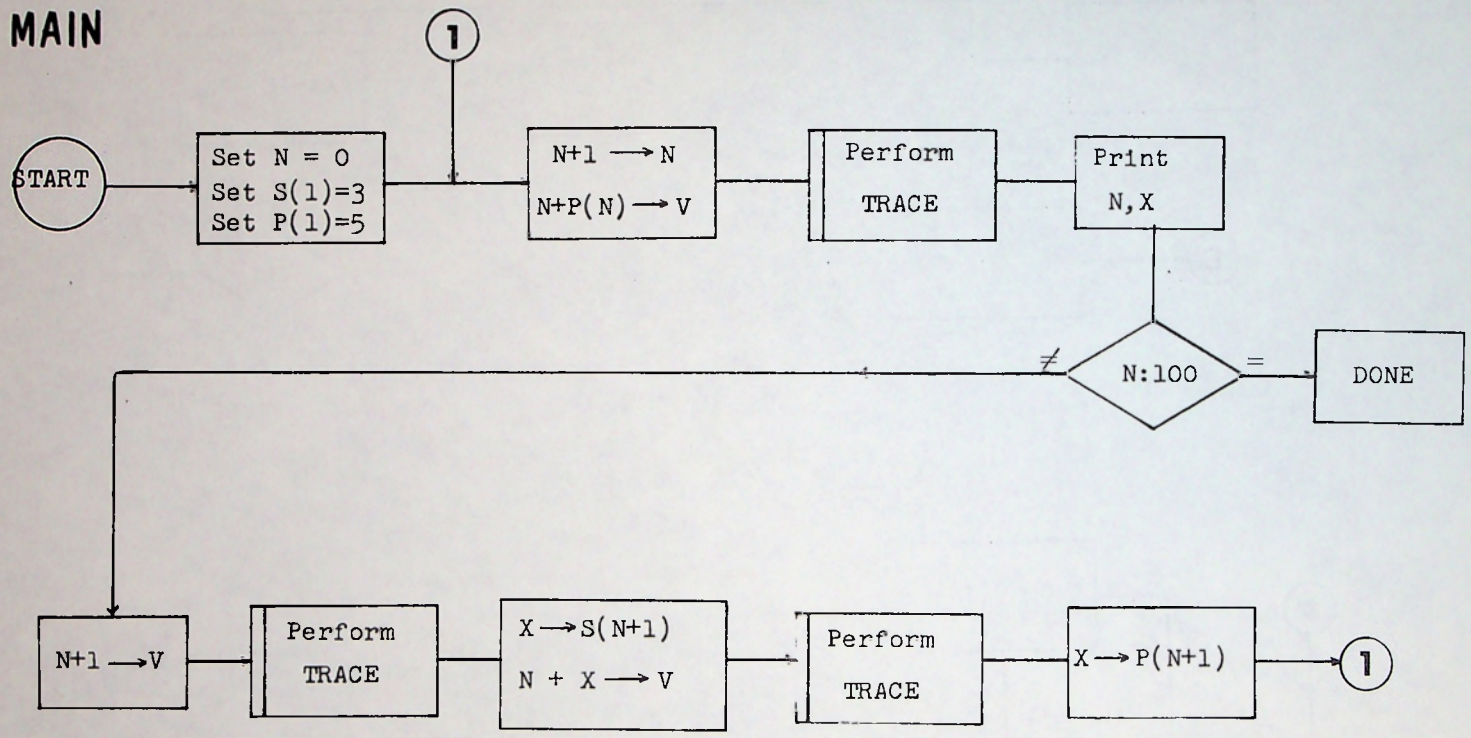
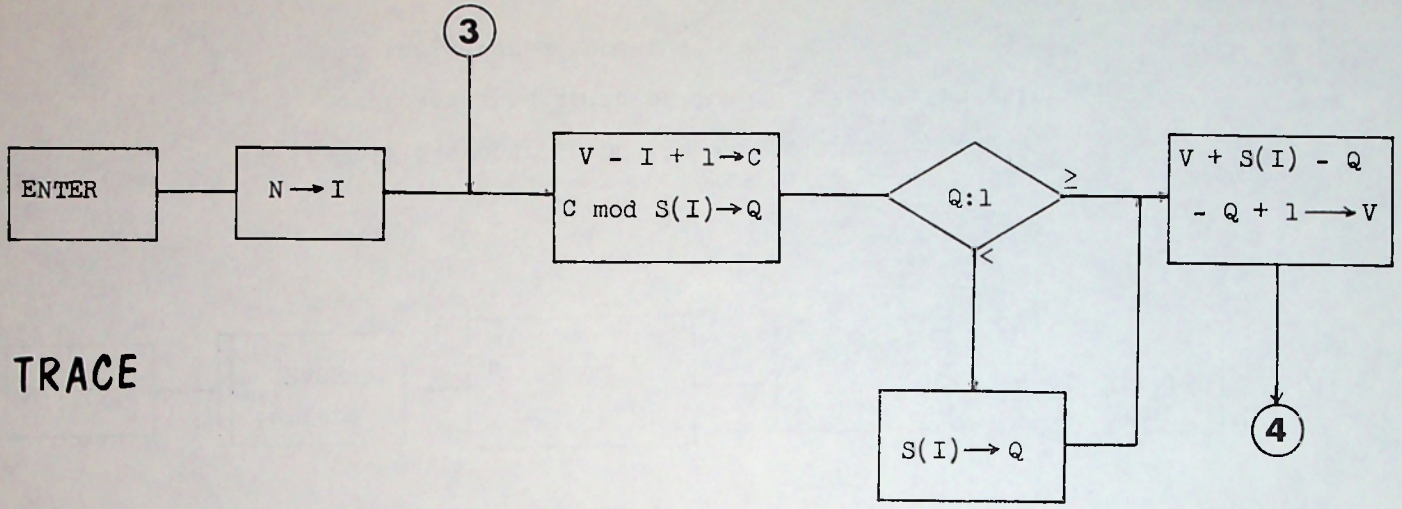


Figure 2. A generalized skeleton of the Figure 1 chart. This aims to highlight certain positional relationships in terms of the variable names used in my solution. The location of any element on the chart can be defined by its column number and cycle number. In that context, cycle number refers to the second line of the cycle, after the reversal phase. The extraction site in Cycle N is seen to be in Column No. $N+P_{(N)}$, i.e. the column number is equal to the cycle number plus the numerical value of the first element (second line) of Cycle N. Similarly, the "starter" value for Cycle N+1 is found in Column No. N+1 of Cycle N, among other locations. The heart of the problem as I see it is determining the numerical value of an element whose location has been defined. My solution does this by tracing the element's path back to the top line of the chart where all values are known.

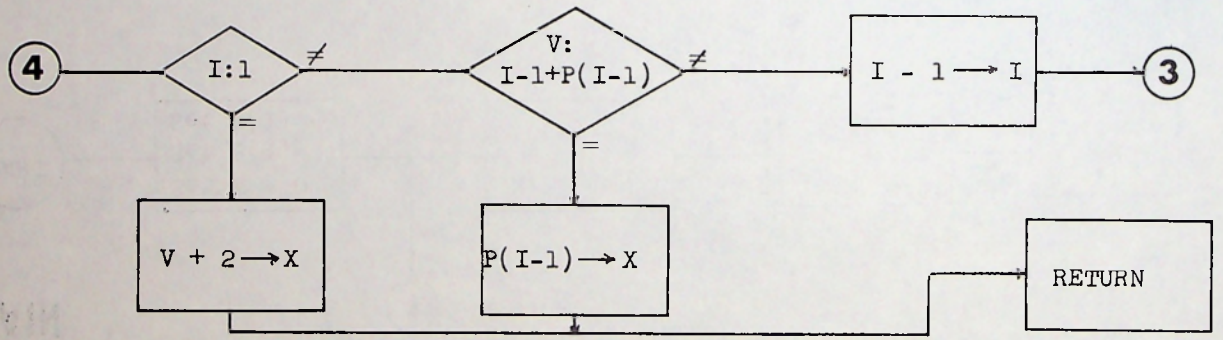
Note: Column $N+P_{(N)}$ is shown here as being to the right of Column $N+S_{(N+1)}$, but it could be to the left, depending on the actual values involved.



This flowchart, and its subsidiary on page 10, expresses the logic of John D. Beeby's solution to the Reverse/Knockout problem.


3
ENTER
N → I
V - I + 1 → C
C mod S(I) → Q
Q:1
≥
<
V + S(I) - Q
- Q + 1 → V
S(I) → Q
4

TRACE


4
I:1
≠
=
V:
I-1+P(I-1)
≠
=
I - 1 → I
3
V + 2 → X
P(I-1) → X
RETURN

I validated my analysis with a BASIC program, which I converted to Fortran in order to use double precision, produce a longer output list, and check the execution speed on a big machine. The Fortran H-Extended venture gave a still longer output list and that was its sole contribution to the project. I am indebted to the College of San Mateo and Stanford University for the privilege of using their computer facilities.

John Beeby's BASIC program

```
10   REM * PROBLEM 212 JOHN D. BEEBY
20   DIM S(100),P(100)
30   N=0
40   S(1)=3
50   P(1)=5
60   N=N+1
70   V=N+P(N)
80   GØSUB 190
90   PRINT " ",N,X
100  IF N=100 THEN 330
110  V=N+1
120  GØSUB 190
130  S(N+1)=X
140  V=N+X
150  GØSUB 190
160  P(N+1)=X
170  GØTØ 60
180  REM * "TRACE" SUBRØUTINE
190  I=N
200  C=V-I+1
210  Q=C-INT(C/S(I))*S(I)
220  IF Q >= 1 THEN 240
230  Q=S(I)
240  V=V+S(I)-Q-Q+1
250  IF I <> 1 THEN 280
260  X=V+2
270  RETURN
280  IF V <> I-1+P(I-1) THEN 310
290  X=P(I-1)
300  RETURN
310  I=I-1
320  GØTØ 200
330  END
```

"Trace" Rationale

If we define the location of some element as Column V, Cycle N and it is in the Qth position of a reversal field of width S(N), then it moved

$$S(N) - (2Q - 1)$$

positions as a result of the Cycle N reversal. I call the above expression the "swing." When evaluated, it indicates how far to the left (-) or right (+) we have to move in order to arrive at the previous location of the element.

If an element is in the extreme right hand position of its reversal field, Q is equal to S(N) and the "swing" is equal to -S(N)+1. That means that the previous location was S(N)-1 positions to the left, and the element was in the extreme left hand position before the reversal.

If an element is in the extreme left hand position of its reversal field, Q is equal to 1 and the "swing" is equal to S(N)-1. We find the previous location by moving S(N)-1 positions to the right which places the element in the extreme right hand position before the reversal.

Of course, if S(N) is odd and Q happens to be equal to (S(N)+1)/2, the "swing" will be zero, placing the element in the same position before and after the reversal.

In general,

$$V(N-1) = V(N) + S(N) - (2Q - 1). \quad *$$

By this reasoning we can trace upward through the chart to reach the top line, but there is one special case to consider at each cycle on the way. The present value in any former extraction site did not arrive there in the usual manner, through a series of reversals. Instead, this value moved in from the "pointer" location of the previous cycle. Thus, when the trace routine lands on an extraction site, the appropriate value can be recalled from array P and the trace is done.

---

*In the programs, this translates to V=V+S(I)-Q-Q+1.

The following names appear in the programs and in Figures 2 and 3.

N is the cycle number.
I is a stand-in for N. It is set equal to N at the beginning of the Trace routine and then decremented during the operation of the routine.
S is an array dimensioned for storing the value of S(N) corresponding to each output item. S(N) is the "starter," the first element in the first line of cycle N. (In the original problem specification this was called the "leader" and designated K.) The value of S(N) defines the width of the reversal field in cycle N.
P is an array dimensioned for storing the value of P(N) corresponding to each output item. P(N) is the "pointer," the first element in the second line of cycle N. It "points" to the extraction site.
V is the column number, referring to the entire chart.
C is the column number within a specific cycle. Cycle N begins in column V, where V is equal to N and C is equal to 1. C is always equal to V-N+1 (or V-I+1) so it is just an intermediate to simplify another expression.

Q is the position of an element within its reversal field. Q refers to the second line of the cycle, after the reversal has taken place. Q is equal to C mod S(N), with the proviso that if Q = 0, then the value of S(N) is assigned to Q.

X is the value returned by the Trace routine in the BASIC program and the flowchart. In the Fortran program as well as the BASIC program it designates the extraction value in the WRITE statement.

The following additional names appear in the Fortran program. They were included to minimize conversions and avoid mixed mode expressions.

Y is the Real*8 equivalent of N.
R is the Real*8 equivalent of I.
U is a constant, 1.0D0 .

Comments on the Programs

The BASIC program was written in Hewlett-Packard Time-Shared BASIC/2000, Level E, and run on the HP-2000 at the College of San Mateo. The output list was limited to 100 items to keep within the number size of the BASIC system. Execution time was approximately 3.5 minutes. The main program or the trace loop or both could be written as FOR...NEXT loops. Here and also in the Fortran program I preferred to use explicitly coded loops. This allowed the BASIC program to conform exactly to the flowchart and also avoided some minor loose ends that seemed to be inherent in the FOR...NEXT approach.

The Fortran program was written initially for the IBM Fortran G compiler and run as a compile-and-go batch job on the IBM System 370/168 at Stanford University. The program had a net count of 62 source statements. With double precision, accuracy problems began to appear around the 250th line of output. The program executed in about 1.95 seconds of CPU time.

The final program was run with the Fortran H-Extended compiler, with the necessary program changes to specify Real*16 for the variables; the compiler optimizing options were not used. The run for 350 lines of output (notice that the solution got to 23-digit numbers) took 36.39 seconds of CPU time.

The problem posed in Contest 15 is intrinsically useless. It seems to be in the nature of the world that it is only such problems, when they are sufficiently challenging, that attract the best minds and the most effort.

The 350 results that Mr. Beeby obtained, then, are not in themselves of any value, but we wish to present them for the record.

1	6	51	2856	101	386162	151	62096664
2	5	52	2148	102	1329220	152	506484280
3	26	53	1206	103	797702	153	204962322
4	14	54	3246	104	1329212	154	163462412
5	16	55	14612	105	821084	155	398156956
6	9	56	2680	106	895228	156	871279960
7	11	57	3992	107	3188714	157	540025168
8	30	58	13592	108	1851050	158	575884716
9	4	59	5002	109	14832	159	1047167076
10	92	60	26202	110	31322	160	2178029746
11	31	61	4568	111	458010	161	4231263224
12	64	62	15446	112	3688224	162	4117932012
13	28	63	33100	113	116162	163	1458630740
14	44	64	20654	114	3166184	164	1195303030
15	46	65	11168	115	2227184	165	2200164448
16	22	66	34240	116	1121388	166	1057643640
17	52	67	27290	117	5578302	167	851838540
18	126	68	21576	118	8772416	168	3119982314
19	256	69	5662	119	8247148	169	4055121086
20	197	70	46388	120	8772622	170	5087750994
21	230	71	69522	121	8247218	171	6560447254
22	110	72	44798	122	8772686	172	17055331578
23	125	73	13012	123	8086144	173	18983964368
24	13	74	44750	124	3251348	174	38249123480
25	21	75	11690	125	4060962	175	64571911298
26	196	76	44828	126	3251062	176	28096848050
27	336	77	29794	127	3556682	177	34244417480
28	684	78	29220	128	14894278	178	4043762140
29	462	79	32324	129	20523970	179	26504033170
30	632	80	29284	130	21792708	180	37033757256
31	852	81	29906	131	43116530	181	86182666702
32	1208	82	83002	132	45675326	182	24865251648
33	780	83	3038	133	71801724	183	59674595214
34	1278	84	26388	134	32044684	184	46679361438
35	544	85	6514	135	51378470	185	156727865474
36	796	86	20036	136	89684842	186	195380136622
37	520	87	18338	137	102304830	187	183843194272
38	8	88	108948	138	342784454	188	234513417442
39	3864	89	130124	139	319528456	189	231375004856
40	98	90	108988	140	6615548	190	244760921936
41	1782	91	161840	141	11450782	191	83566681700
42	5316	92	116830	142	6614898	192	288741029076
43	6372	93	274270	143	11406830	193	658615695470
44	7714	94	225538	144	16197704	194	973492533902
45	7274	95	21786	145	74724060	195	1197324615328
46	3818	96	354738	146	150807696	196	1160943169758
47	4254	97	452492	147	217464252	197	939909285014
48	9046	98	353666	148	135382098	198	816942516594
49	6412	99	456634	149	7301960	199	1700983282472
50	2168	100	308230	150	43535338	200	873368821832

The 350 results to the Reverse/Knockout problem obtained by John D. Beeby.

201	1647940441800	251	9432599231110202	301	27010325179975097 28
202	629077431134	252	22008829507645552	302	248813610062368090
203	1515431095038	253	13942311078213494	303	3170761985355947130
204	1094423372758	254	5713338879885968	304	4514074813577561646
205	439697243988	255	5024762207463218	305	2089422656356836610
206	448533959454	256	15328397915860412	306	3945961824221695744
207	2765040074696	257	2114534198022979 0	307	2640728795210343944
208	1588989792664	258	344363647420096 60	308	6115597430786987116
209	2930116079276	259	870516323968750	309	1283827093950946 7840
210	2724480645010	260	3393279985709486	310	1710649800696899 5546
211	2170128551744	261	6560274044314209 8	311	2255756075289892 3366
212	795478898464	262	3393279985711506	312	3478014592624742 8694
213	1618908577662	263	6560274044310440 8	313	3012801568762345 7564
214	6899877343738	264	3313177760024105 6	314	3606081538876500 1596
215	922798044556 0	265	291900783555120	315	6441269783695772 2910
216	689987734604 4	266	715530633647864	316	3606081538876500 1814
217	922797940031 8	267	1203742010657640 2	317	6441269783695750 2954
218	1554959674543 4	268	2644820534948944 8	319	7946114803538837 3144
219	61693908003 16	269	689798507574723 4	319	9550846357892301 6408
220	13054046894786	270	4656213632842235 8	320	1168233219120636 16228
221	39167901725720	271	1084342615414151 10	321	2569117494125808 85860
222	13054046916796	272	4656213632842232 2	322	1168233219120636 15640
223	39167901696180	273	1084342615414052 34	323	2569117494125811 28528
224	7463291813573 4	274	172232814942016 6	324	3115252361992245 53570
225	37597628692670	275	31928502686558550	325	2012723672822564 02698
226	10033206912682	276	491832961881654 52	326	3393972001665400 65576
227	564443940069 90	277	114859371914710450	327	7966379202883551 1890
228	10033206912568	278	242419096979095600	328	6927999118541990 35568
229	56445261921258	279	155709757051246944	329	1101908320803391 63898
230	14409638794316	280	154905443618361306	330	3240875925060103 18248
231	41390590285006	281	288426465173612734	331	4329104932742925 39656
232	5486961381519 8	282	427585929781267356	332	2825598450897335 46496
233	1490372640190 94	283	381290131522582226	333	7626294075773757 8554
234	445438591982 58	284	597207492972013896	334	8986150543975403994
235	156183522614262	285	548559259159863400	335	1303682061781492 24508
236	3346764317568 40	286	37951257850408736	336	5007678163999435 81036
237	4970023897580 36	287	432272077613003 04	337	6377180779053562 77556
238	803243570041582	288	126610163038726794	338	1528958239797280 784844
239	1367520555923724	289	219775838337964826	339	2078783932486557 972454
240	1689166854660282	290	121246894284242046	340	1528958239797280 782846
241	2764676536466134	291	9793095217665970	341	2078783932486558 417254
242	2266659858104154	292	8622803322469070	342	1600898959556427 576820
243	915251093748280	293	35475875273817654	343	1016754307475582 42914
244	1299448425668730	294	7463931371303714 46	344	9353797429191909 26484
245	3986391157615022	295	6553631486825824 50	345	1188187750216303 725622
246	1299448425668858	296	7463931371303715 08	346	4792188315125703 600100
247	3986391154873888	297	6553631486825854 26	347	6145767301356526 647146
248	8717089329013488	298	5649154275749290 50	348	7176894772488398 343394
249	8672193321601416	299	1154435447905094 504	349	1254089344285417 7111280
250	8756264978594208	300	1532300970211599934	350	5188797364922446 080176

In issue 61 (April) we presented a plan devised by Associate Editor David Babcock to produce, on demand, patterns of letters that contain hidden words. The Fortran program is given here.

```
      DIMENSION IPTR(100),IWORD(20,100),LEN(100),LETTER(26),
     +          MATRIX(42,42)
C
C     INITIALIZE CHARACTER VARIABLES
C
      DATA IAST/1H*/,IBLANK/1H /,LETTER/1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,
     +     1HI,1HJ,1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,1HW,
     +     1HX,1HY,1HZ/
C
C     READ ARRAY SIZE AND INITIALIZE PUZZLE MATRIX
C
      READ(5,500)N
      IF(N.LT.1 .OR. N.GT.40)GOTO 150
      AN=N
      N1=N+1
      N2=N+2
      DO 10 I=2,N1
      DO 10 J=2,N1
   10 MATRIX(I,J)=IBLANK
      DO 20 I=1,N2
      MATRIX(I,1)=IAST
      MATRIX(1,I)=IAST
      MATRIX(I,N2)=IAST
   20 MATRIX(N2,I)=IAST
C
C     READ WORD LIST AND COMPUTE WORD LENGTHS
C
      DO 50 M=1,100
      READ(5,510)(IWORD(I,M),I=1,20)
      IF(IWORD(1,M).EQ.IBLANK)GOTO 60
      DO 30 I=2,20
      IF(IWORD(I,M).EQ.IBLANK)GOTO 40
   30 CONTINUE
      I=21
   40 LEN(M)=I-1
   50 IPTR(M)=M
C
C     SORT WORD LIST IN LENGTH ORDER
C
      M=101
   60 M=M-1
      IF(M.EQ.0)GOTO 150
      L=M-1
   70 K=0
      IF(L.EQ.0)GOTO 90
      DO 80 I=1,L
      IP1=IPTR(I)
      IP2=IPTR(I+1)
      IF(LEN(IP1).LE.LEN(IP2))GOTO 80
      K=I-1
      IPTR(I)=IP2
      IPTR(I+1)=IP1
   80 CONTINUE
      L=K
      GOTO 70
C
C     COMPUTE RANDOM MATRIX LOCATION FOR ALL WORDS
C
   90 DO 120 I=1,M
  100 IX=INT(AN*RANDOM(0))+2
      IY=INT(AN*RANDOM(0))+2
      IDX=INT(3.0*RANDOM(0))-1
      IDY=INT(3.0*RANDOM(0))-1
      ITX=IX
      ITY=IY
      LENT=LEN(I)
```

```
C
C       CHECK IF WORD FITS
C
        DO 110 J=1,LENT
        ICHAR=MATRIX(ITX,ITY)
        IF(ICHAR.NE.IBLANK .AND. ICHAR.NE.IWORD(J,I))GOTO 130
        ITX=ITX+IDX
  110   ITY=ITY+IDY
C
C       PUT WORD IN PUZZLE MATRIX
C
        DO 120 J=1,LENT
        MATRIX(IX,IY)=IWORD(J,I)
        IX=IX+IDX
  120   IY=IY+IDY
C
C       FILL BLANK SQUARES WITH RANDOM CHARACTERS
C
        DO 130 I=2,N1
        DO 130 J=2,N1
        IF(MATRIX(I,J).NE.IBLANK)GOTO 130
        ICHAR=INT(26.0*RANDOM(J))+1
        MATRIX(I,J)=LETTER(ICHAR)
  130   CONTINUE
C
C       PRINT PUZZLE MATRIX
C
        DO 140 I=2,N1
  140   WRITE(6,520)(MATRIX(I,J),J=2,N1)
C
C       END OF RUN
C
  150   CALL EXIT
C
C       FORMAT STATEMENTS
C
  500   FORMAT(I2)
  510   FORMAT(20A1)
  520   FORMAT(/1X,40(A1,2X))
C
        END
```

1.11 Set c = 1/360.
1.12 Set n = 0.
1.13 Set m = 1.
1.14 Set t = 0.
1.21 Set n = n+1.
1.22 Set m = m+1.
1.23 Set t = t + (n/m).
1.24 Set d = |t - ip(t+.5)| .
1.25 To step 1.21 if d ≥ c.
1.31 Display n,t,d.

The Affectionate Racer
Continued from page 2.

n = 82.
t = 77.997932
d = .002068

Further calculations revealed some of the critical values for the parameter, c, as shown in this table. The next critical value would require very high precision arithmetic.

c	n	t	d
.5	2	1.1666666	.1666666
.1	3	1.9166666	.0833333
.08	9	7.0710318	.0710318
.07	10	7.9801227	.0198773
.01	29	26.0050129	.0050129
.005	82	77.997932	.002068
.002	225	220.000039	.000039
.00003	4548	4540.00001	.0000095

We begin here a new feature: homework assignments for computing students. Each assignment has been debugged and time-tested through many semesters of use. Most of the loopholes have been plugged, so that a student should know what to do from the directions given here.

homework ①

Differencing

Given a deck of cards. Each card bears one number--a single value of some function.

Draw a flowchart for the following logic: after reading each card, print on one line the functional value and the first, second, and third differences.

The length of the deck is unknown, but you can assume (since we want to go to third differences) that it is at least 10 cards long. The end of the deck is detected by the simplest of end-of-file procedures; namely, that there are no more cards to be read.

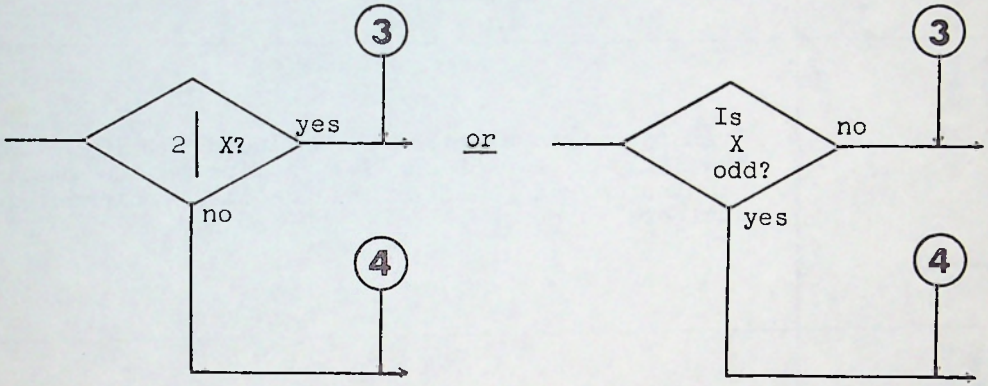
For an elegant solution to this problem, you could spend a lot of time worrying about not developing or printing various differences on the first three cards, since they will be garbage anyway. You are to ignore this fact; the garbage can be crossed off the listing with a pen. If that sounds like sloppy computing, keep in mind the dictum "First make it work--then make it pretty." We are concerned here with the first part of the dictum. For now, we want only to get the differencing logic expressed.

When you have completed your flowchart (which should be fairly simple), consider the following points:

Have you made any assumptions about the data, such as its order, or whether the numbers are all positive? (There should be no assumptions.) Does your flowchart produce differences, as asked for, or have you produced the absolute values of the differences? How much storage would your logic require? Will it still work if the data deck is 100,000 cards long? Have you identified all your variables? Does your flowchart require a personal lecture to go with it, or could anyone understand it? This is intended to be a simple, straightforward task--don't make it into something grandiose.

ODD & EVEN

An integer is stored at address X. We have come to the point in a problem solution where our course of action depends on whether the number at X is odd or even; that is, to the situation expressed on a flowchart as:



The direct and natural thing to do is divide X by 2 and test the remainder of that division for zero (and if zero, X is even, and we go to reference 3). The trouble with that is

1. division is a costly operation in terms of CPU time, and
2. for most coding languages, it is messy to extract the remainder of a division.

There is, of course, the MOD function in many languages which does what we want, but how does <u>it</u> get implemented at the machine language level?

Similarly, some languages have op-codes for JUMP ON ODD but again--how do <u>they</u> work?

Try another tack. Perform an AND between word X and the number <u>one</u>. This operation will mask in the unit's bit of X in the accumulator, so that a JUMP ON ZERO will be taken if X is even.

There must be many other ways...

PC62-20

GENERATIONS

I 1950–57	Vacuum tube machines. Early models one-of-a-kind. First mass produced machine the Univac I. First, drum and CRT storage; cores on later models. Included IBM 650 and 704. Decimal machines for business; binary for scientific. Reliability relatively low.
II 58–64	Transistorized machines. Included IBM 7090, 1620, and 1401. LGP-30 and G-15 dominated the small machine market. Still had decimal equipment for business purposes.
III 64–70	Byte logic, and 1/2" chip circuitry, typified by the System 360. Now all machines are binary. The 16-bit minis began around 1968.
IV 70–77	Floppy disk control, typified by the System 370; for the first time, identical machines were operating in the field. 8-bit micros began in 1973, and the personal and hobby machines started in 1975. Rise of virtual storage.
V 77.....	The watchword is <u>firmware</u>, as more and more of the burden of overhead is shifted from software to hardware. 50 nanosecond complete cycle times, and going down. Still wedded to magnetic cores, at least at the start of this generation. Powerful systems being built around microprocessors.